– SAFAPS SIM –
ARCHITECTURE DOCUMENT

Jeremy Harrault

SWORDFISH

Objectives of this document

The purpose of this document is to present the architecture of the SAFAPS SIM project. It will contain diagrams as well as explanations to describe the architectural choices in order to fulfil the requirements. The information contained in the document act as a guide in order to fully develop, deploy and setup SAFAPS.

Glossary and Terminology

– A –

API: Application Programming Interface

– J –

JSON: JavaScript Object Notation

– R –

REST: Representational State Transfer

– S –

S&F: Stress and Fatigue

SAFAPS: Stress and Fatigue Audit and Prediction Service

Document Description

Title	SAFAPS SIM : Architecture Document	
Creation date	25/01/2016	
Publication date	26/01/2016	
Product Owner	Jeremy Harrault	hajr15bp@ju.se
Authors	Jeremy Harrault	hajr15bp@ju.se
	Saqib Ahmed	ahsa1511@student.ju.se
Subject	Architecture Document	
Model version	1.0	
Document version	0.9	

Revisions table

Date	Rev.	Author	Modified Section(s)	Comments
25/01/16	0.1	Jeremy Harrault	All	Add empty sections.
28/01/16	0.2	Jeremy Harrault	3.	Add context and database view.
29/01/16	0.3	Jeremy Harrault	3.	Add invoice table in database view and add additional information.
02/02/16	0.4	Jeremy Harrault	1. 3.	Remove "Introduction and Management Summary" part. Add the general architecture principals. Add standardization from information viewpoint.
04/02/16	0.5	Jeremy Harrault	3 .	Explanation "datetime". Modify Database standardization rules
05/02/16	0.6	Jeremy Harrault	3.	Move foreign key between Evaluation and Result into Evaluation table. Add section for status of S&F evaluation Add "status" column in database to "evaluation".
05/02/16	0.6.1	Jeremy Harrault	1. 3.	Show the asynchronous process in Global architecture diagram. "S&F Evaluator" => "S&F Evaluator Service"
09/02/16	0.7	Jeremy Harrault	1. 2. 3.	Describe the goal of SAFAPS SIM Add architectural decisions and their advantages Add functional viewpoint
12/02/16	0.8	Jeremy Harrault	3.	Add Symfony2 file system explanations Add deployment diagrams
20/02/16	0.9	Saqib Ahmed	2. 3.	Add RabbitMQ description RabbitMQ installation details

Table of Contents

List of Tables

List of Figures

1. General Architecture Principles

1.1. Web application

SAFAPS SIM is a Web application that can be reached using HTTP. It includes an API that handle the business to be performed. This API can be called by any HTTP client; that is either humans directly requesting the server from a browser for instance or other systems which need Stress and Fatigue information.

Beside the API which returns raw data, SAFAPS SIM can also return these data wrapped in a user interface to be displayed onto a browser and become more human readable.

The SAFAPS SIM API is compliant with the REST architecture.

1.2. Global Architecture

The architecture of SAFAPS SIM presents distinct layers, each of them fulfilling their own tasks.

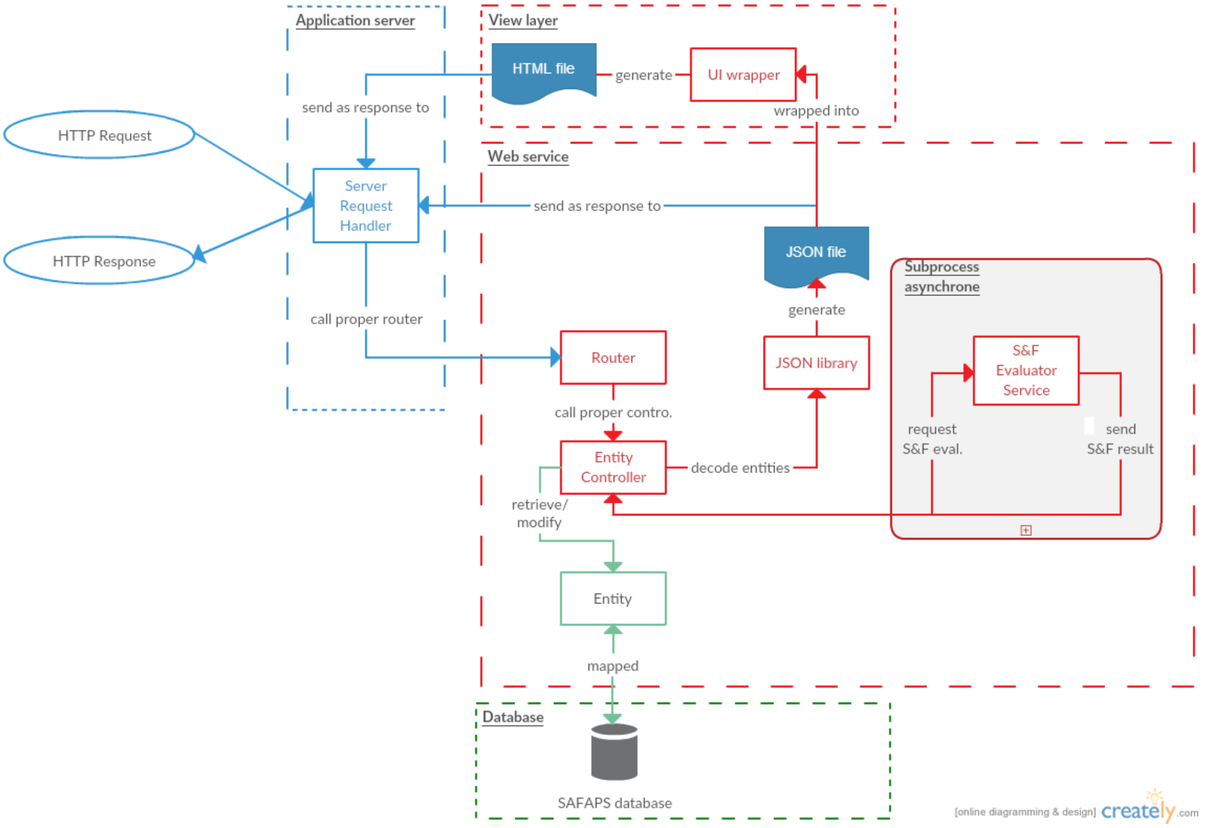


Figure 1.1: SAFAPS SIM global architecture

Layer	Description
Application server	The application server is first layer that the HTTP request goes across. This layer is between the software application and the operating system that the web server is running on. It catches the HTTP requests received by the server and pass it to web service.

Layer	Description
Web service	The web service includes all business logic that will be computed. It includes the router which will call controllers depending on the resource pointed by the HTTP request. Controllers can then access and modify values within entities and call the S&F evaluator by launching an asynchronous process. The input and output handled by the web service are formatted in JSON.
Database	The layer is in charge of storing data.
View layer	The view layer is used to lay out the data returned by the web service within HTML to present data onto browser for example and therefore become more human readable.

Table 1.1: SAFAPS SIM global layers

2. Architectural Design Decisions

2.1. Rest Architecture

The core of SAFAPS SIM is a RESTful web service. REST uses simple HTTP as the communication protocol between the client and the server.

Using HTTP already defines of way of performing requests with URL and HTTP methods.

Unlike SOAP, REST does not need to send the format of the response returned by the server. It facilitates the call to the web service on the client side since it does not need any libraries or extra component to decode the response.

Therefore, testing a REST architecture can be done very easily using a browser.

2.2. PHP

SAFAPS SIM is coded in PHP. This language offers different advantages:

- Open-source language: It makes the support and the extensibility of the language easier considering the huge community of PHP developers.
- Good documentation: Besides the official PHP documentation which is very complete, a lot of technical trouble can be resolve after a search on internet.
- Easily deployed: PHP is a scripting language which does not require any compilation. The PHP file only needs to be placed into the server directory.
- Optimized for building web applications: PHP natively provides tools and features like accessing method and URL of the requests as so that the development of
- Well-known language: A lot of people are using PHP. Then, it is easier to find programmers PHP skills who can handle the project.

2.3. Symfony2 Framework

The codebase of SAFAPS SIM is running with Symfony2 framework. This choice has been made many reasons:

- Premade skeleton: Symfony2 pre-define the architecture basics of the web application.
- Saving time: Using a framework is a gain of development time since a big part of the work is already coded. The developers can focus on the business part of the application.
- Already experienced team: The development team is already familiar with this framework, reducing the learning phase.
- Extensible: Some “bundles” can be plugged to Symfony2 if need be, adding then new features to the system to be built either at runtime or at the development time.

2.4. RabbitMQ

RabbitMqBundle is used with Symfony2 Framework in order to incorporate messaging in SAFAPS. The aim of using this bundle is to let the message queue provided by RabbitMQ handles the events, which can be easily controlled by Symfony2. We have choose RabbitMQ due to following reasons:

- Events are stored in queues independently and contains all the information for processing.
- Processing of events is faster because it is possible for multiple processes to read from a queue.

- Queues can be mirrored across many machines, ensuring the backup in case of hardware failure.
- SAFAPS would be more responsive because time consuming requests can be processed asynchronously.

2.5. S&F Evaluator Sub-process

In order to avoid the client to be blocked waiting for the server delivering the response to the S&F request, SAFAPS SIM will execute these kind of request asynchronously.

For that purpose, a parallel process in charge of executing S&F request will be running when an S&F request is submitted. Once the result generated, it will be sent back to the client.

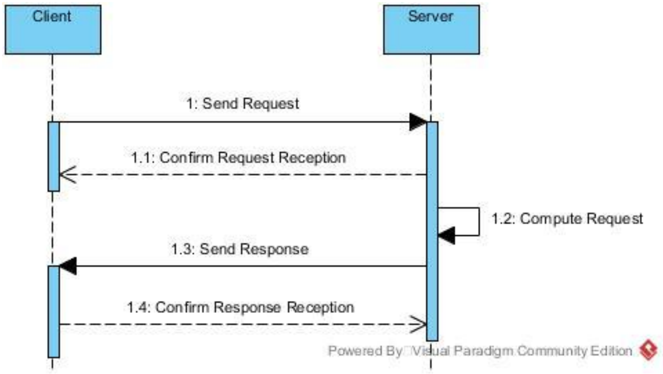

Client
Server
1: Send Request
1.1: Confirm Request Reception
1.2: Compute Request
1.3: Send Response
1.4: Confirm Response Reception
Powered By Visual Paradigm Community Edition

Figure 2.1: Asynchronous call for S&F request

3. Viewpoints and Views

3.1. Context Viewpoint

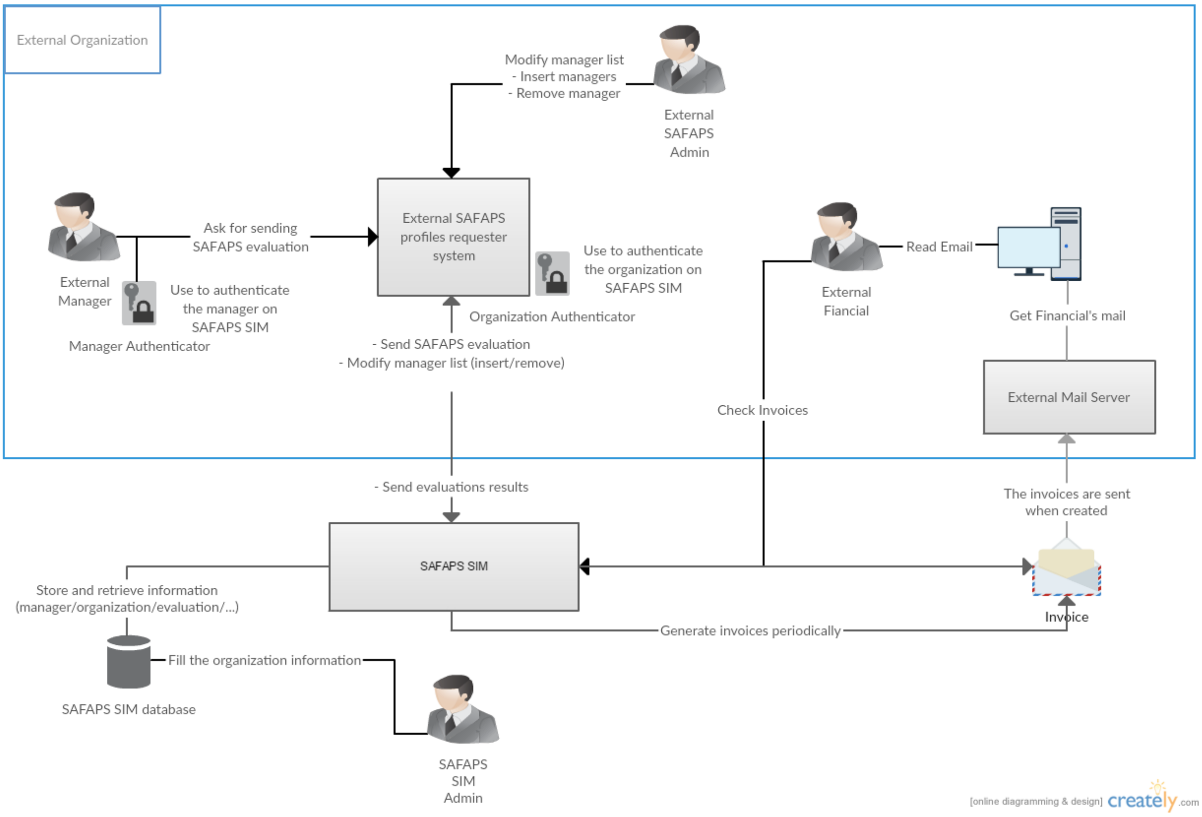
SAFAPS SIM is a system that is mainly used by external systems from external organizations. Managers from such organizations can use SAFAPS through their own internal system. For security concerns, managers get authenticated by SAFAPS SIM using their unique authenticating keys. Managers are not supposed to request SAFAPS SIM directly.

Once the evaluation has been performed by SAFAPS SIM, it will be sent back to the organization's system.

When managers are added to or removed from the organization's system, it notifies SAFAPS SIM so that the new managers are added to or removed from SAFAPS SIM too.

The SAFAPS SIM system is not free of charge and organization using SAFAPS must pay for it. Periodically, invoices are automatically sent by SAFAPS SIM to organizations. External organizations' financials can consult such invoices for the organization in two ways. They either can consult them from their mail or directly from SAFAPS SIM website.

SAPAFS SIM is a simulation software. To limit the work to do on the back-end of SAFAPS website, the information about the organizations is filled in SAFAPS SIM by the SAFAPS administrators.



External Organization
Modify manager list
- Insert managers
- Remove manager
External SAFAPS Admin
Ask for sending SAFAPS evaluation
External SAFAPS profiles requester system
Use to authenticate the organization on SAFAPS SIM
External Manager
Use to authenticate the manager on SAFAPS SIM
Manager Authenticator
Organization Authenticator
- Send SAFAPS evaluation
- Modify manager list (insert/remove)
External Fiancial
Read Email
Get Financial's mail
External Mail Server
Check Invoices
- Send evaluations results
The invoices are sent when created
SAFAPS SIM
Store and retrieve information (manager/organization/evaluation/...)
Invoice
Generate invoices periodically
Fill the organization information
SAFAPS SIM database
SAFAPS SIM Admin
[online diagramming & design] creately.com

Figure 3.1: Context diagram

3.2. Functional Viewpoint

3.2.1. Functional parts

SAFAPS SIM's API's architecture presents 4 main parts. Each of these parts are described in the following subsection.

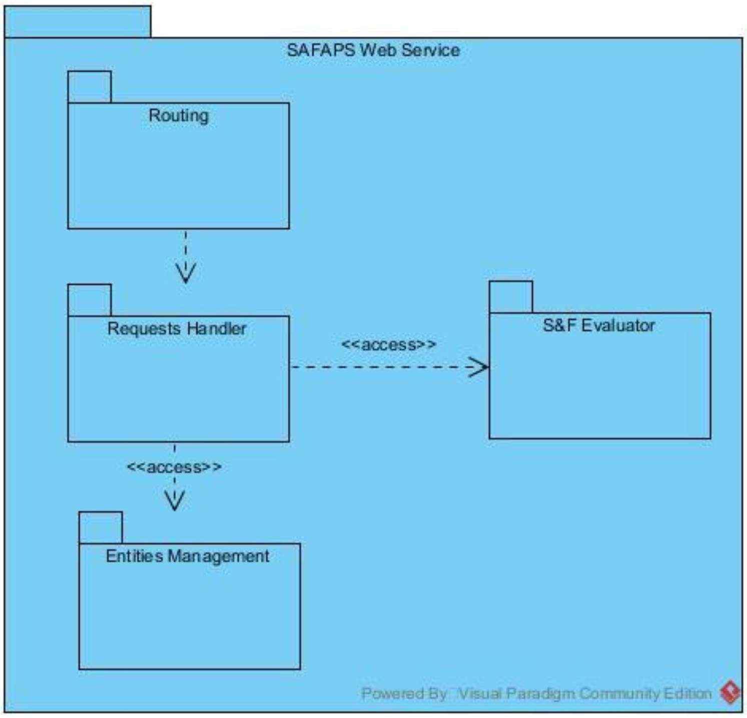
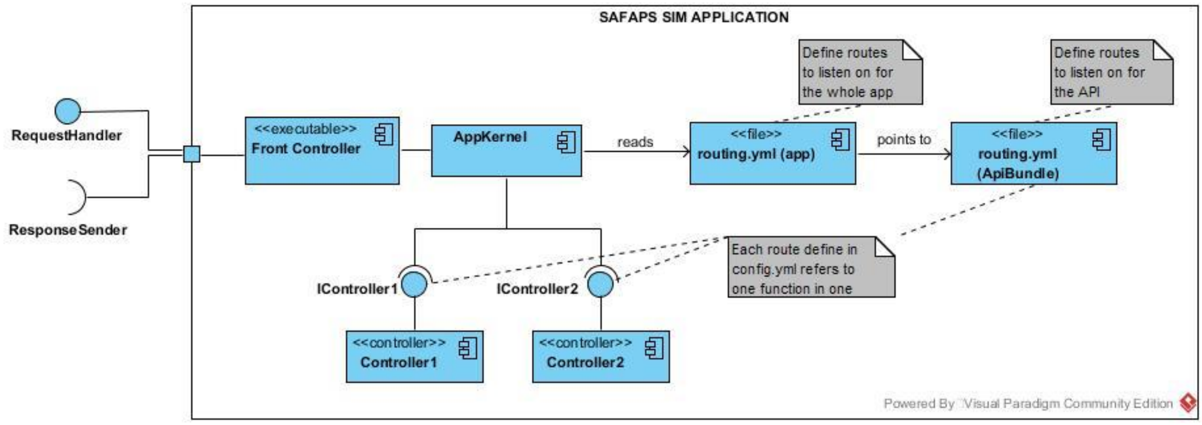


Figure 3.2: Package diagram

3.2.2. Routing

When a client requests SAFAPS SIM, the first task to be done is routing the request. It consists in calling the proper function according to the resource and the method of the HTTP request.

This part is mainly handled by components set up by Symfony2.



SAFAPS SIM APPLICATION
Define routes to listen on for the whole app
Define routes to listen on for the API
RequestHandler
ResponseSender
<<executable>> Front Controller
AppKernel
reads
<<file>> routing.yml (app)
points to
<<file>> routing.yml (ApiBundle)
Each route define in config.yml refers to one function in one
IController1
IController2
<<controller>> Controller1
<<controller>> Controller2
Powered By Visual Paradigm Community Edition

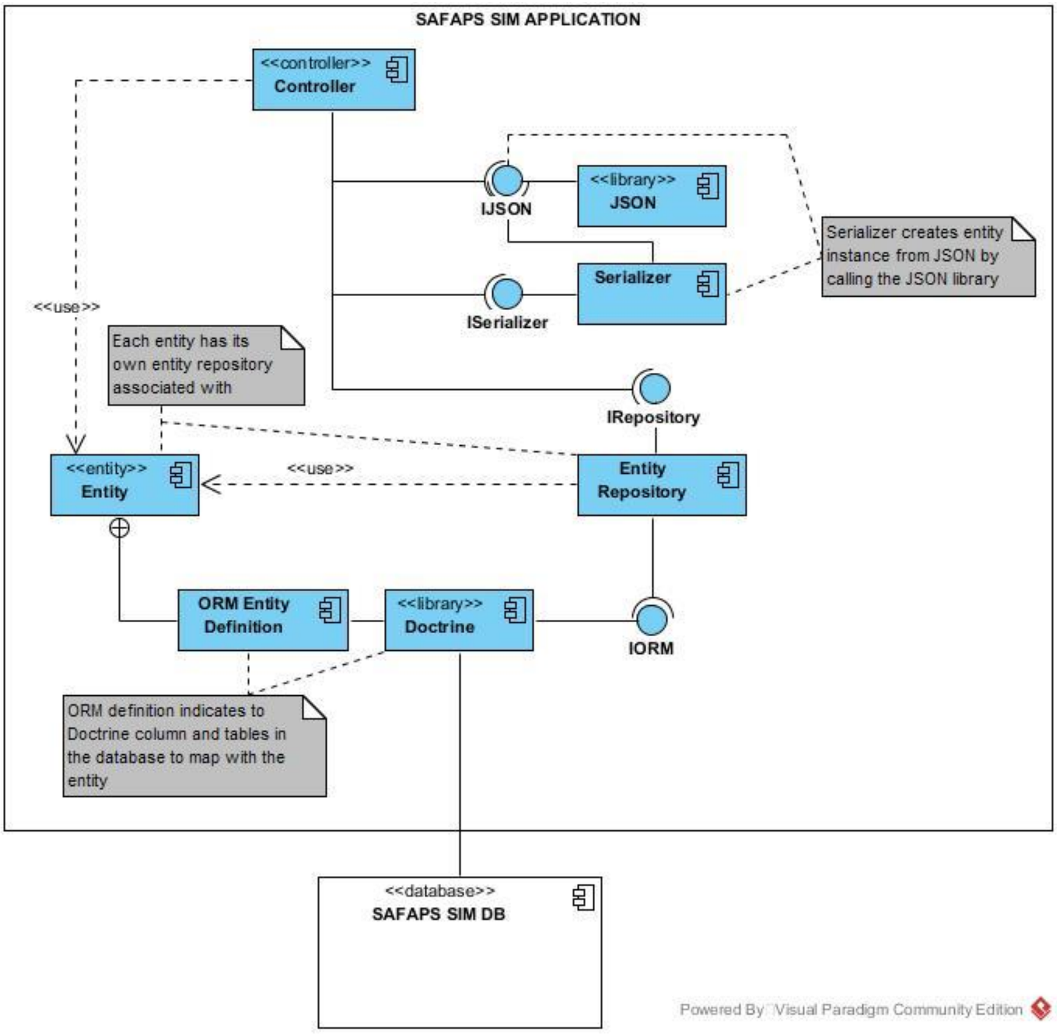
Figure 3.3: Component diagram for routing requests

Component	Description
Front Controller	It is first script that is called after receiving the request. It sets the global environment settings (production, debug, testing)
App Kernel	It loads all Bundles needed by the application to work. It also calls the proper controller based on the resources and the routing file.
routing.yml	The routing files define which controller should be called according to the resource pointed in the request.
IControllers	These interfaces are built from the routing.yml files. Indeed, they include functions to be implemented according to which functions have been defined as to be called by the routing.yml
Controllers	They are classes implementing the IControllers interfaces.

Table 3.1: Component descriptions for routing request

3.2.3. Requests Handler and Entities Management

The process of executing the request, retrieving, creating or modifying entities is conducted by the controllers. They constitute the central part of the entities management by calling different components to get entities, modify their attribute and store them to the database.



SAFAPS SIM APPLICATION
<<controller>>
Controller
IJSON
<<library>>
JSON
Serializer creates entity instance from JSON by calling the JSON library
Serializer
ISerializer
<<use>>
Each entity has its own entity repository associated with
IRepository
<<entity>>
Entity
<<use>>
Entity Repository
ORM Entity Definition
<<library>>
Doctrine
IORM
ORM definition indicates to Doctrine column and tables in the database to map with the entity
<<database>>
SAFAPS SIM DB
Powered By Visual Paradigm Community Edition

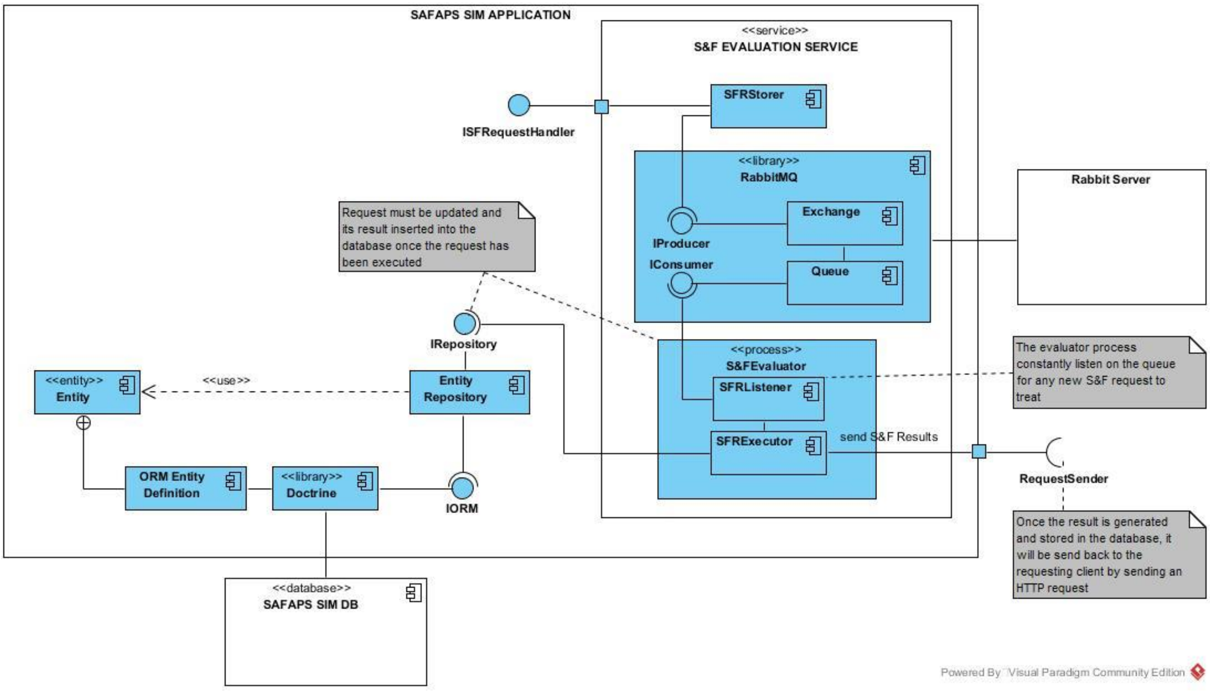
Figure 3.4: Component diagram for managing entities

Component	Description
Controller	The controller is the entry point to treat the HTTP request body. It gets the HTTP request body and, depending on the controller which is called, perform operations calling other components like libraries or even other controllers.
JSON and Serializer	These libraries are used to transform the request body in JSON to entities or entities to JSON objects.
Entity Repository	It is used to perform operations on entities in the database.
Entity	The entity are the objects that can be manipulated by the controllers to get or change values inside.
ORM Entity Definition	The ORM definition defines le mapping between the entities as classes and as database tables. The attributes within classes are to be mapped with column in the database.
Doctrine	It is a component of Symfony2 sending request to the database and using the ORM definition of the entities.

Table 3.2: Component descriptions for entities management

3.2.4. S&F Evaluator

Once an S&F request is received and registered in the database by controllers, the request is pushed into the S&F Evaluation Service. It is in charge of treating the S&F request, store the result in the database and



SAFAPS SIM APPLICATION
<<service>>
S&F EVALUATION SERVICE
SFRStorer
ISFRequestHandler
<<library>>
RabbitMQ
Exchange
Queue
IProducer
IConsumer
Rabbit Server
Request must be updated and its result inserted into the database once the request has been executed
IRepository
<<process>>
S&FEvaluator
SFRListener
SFRExecutor
send S&F Results
The evaluator process constantly listen on the queue for any new S&F request to treat
<<entity>>
Entity
<<use>>
Entity Repository
ORM Entity Definition
<<library>>
Doctrine
IORM
RequestSender
Once the result is generated and stored in the database, it will be send back to the requesting client by sending an HTTP request
<<database>>
SAFAPS SIM DB
Powered By Visual Paradigm Community Edition

Figure 3.5: Component diagram for executing S&F requests

Component	Description
SFRStorer	It is called when a new S&F request is to be treated. It stores these requests to a queue, awaiting to be treated.
RabbitMQ Server	It is the server where the S&F requests and their associated status is stored.
RabbitMQ	RabbitMQ is a Symfony2 bundle providing the interfaces to the web application to communicate with RabbitMQ server.

Component	Description
S&FEvaluator	This component represents an asynchronous process dealing with the S&F request to treat. As long as the web application is running, this process will always be running.
SFRListener	It constantly listens on RabbitMQ queue to get any incoming S&F request. Once a request is picked up, the listener turns in pause until the result of the S&F request under evaluation is done.
SFRExecutor	It retrieves data from the database and compute them to generate the result of the S&F request. Once done, the result is stored in database and sent in an HTTP request to the URL contained in the request

Table 3.3: Component descriptions for S&F request evaluator

3.2.5. Overview

Below is the diagram showing all aggregated parts which have been defined in the previous sections.

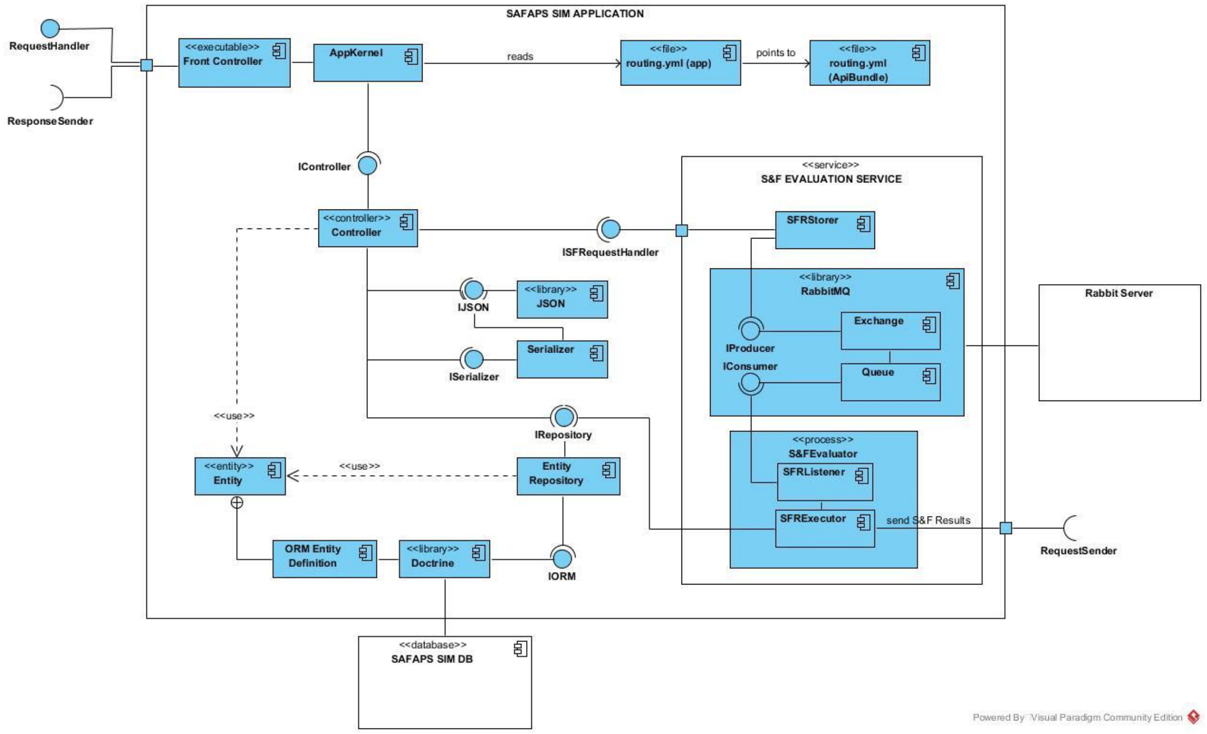


Figure 3.6: Component diagram of SAFAPS SIM

3.3. Information Viewpoint

3.3.1. S&F Evaluation Request States

The S&F requests go through different states from the creation to the point of they are actually performed.

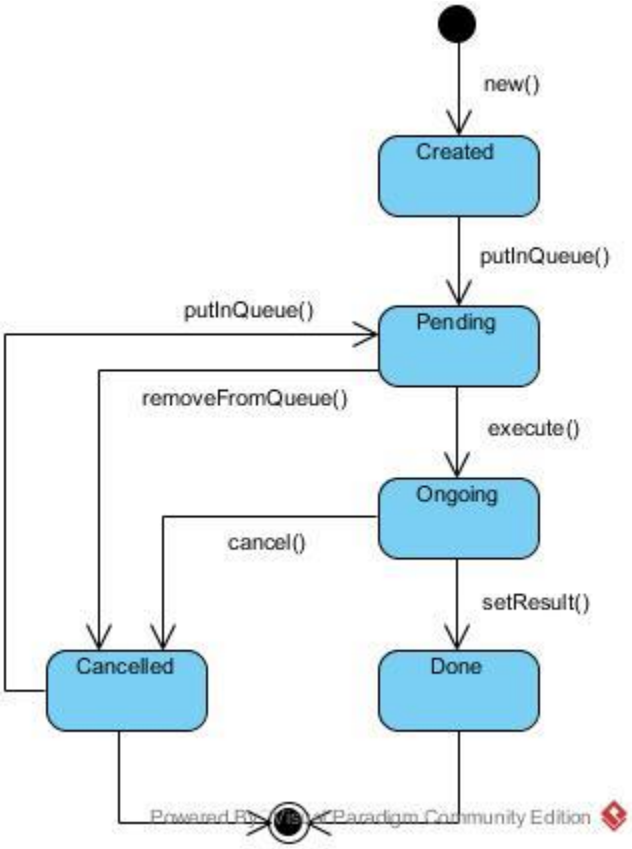


Figure 3.7: State Chart of the S&F evaluation request

State name	Description
Created	When the evaluation request is instantiated and inserted into the database, its status is created.
Pending	When the evaluation request is put in the queue awaiting to be treated by the S&F evaluator service.
Ongoing	When the evaluation request is taken from the queue in order to be treated by the S&F evaluator service.
Done	When the result of the S&F evaluation request is set and inserted inside the database.
Cancelled	When the S&F evaluation request has been remove from the queue or the execution of the request has been interrupted.

Table 3.4: Description of S&F evaluation request states

3.3.2. Database View

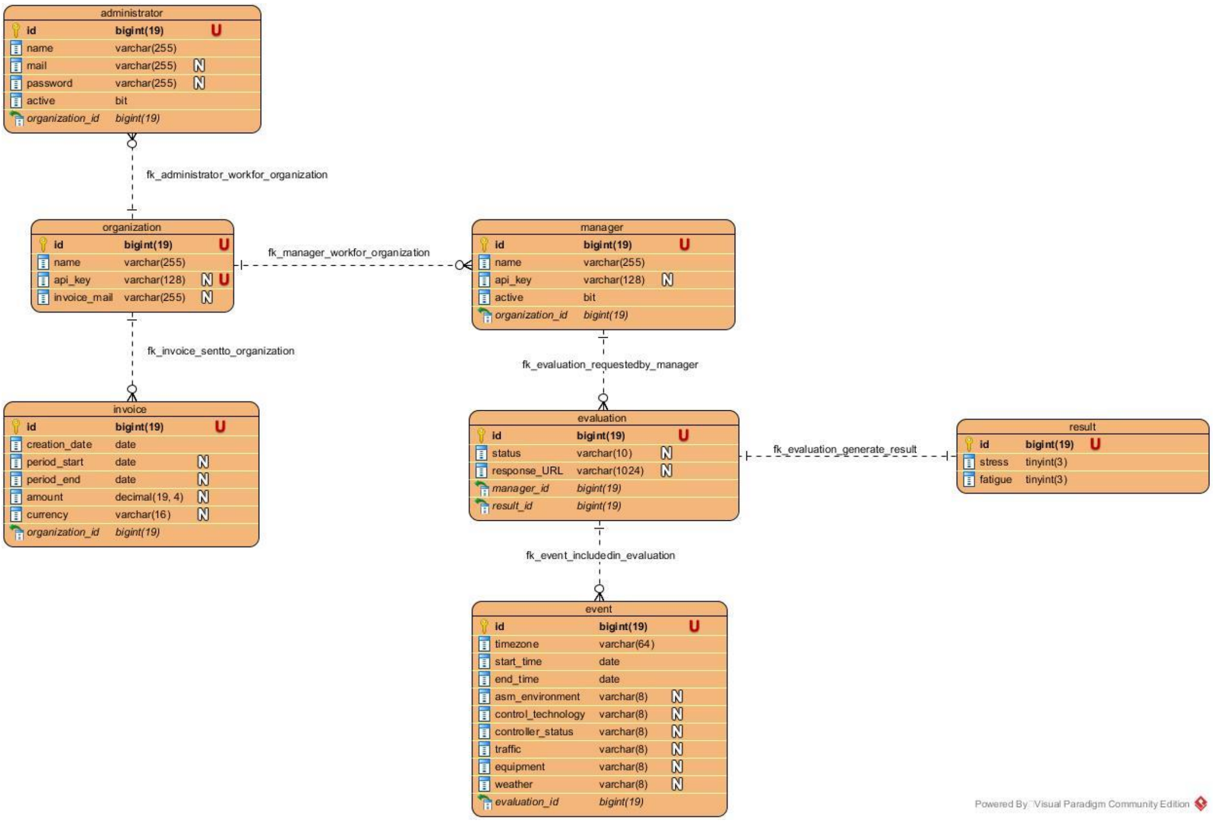


Figure 3.8: Database entity diagram

Table	Column	Additional information
- administrators - managers	- active	The active field represent whether the account is still authorized to use SAFAPS SIM functionalities. The type of this field represent a data with only 2 exclusive possible values. Depending on the database implementation, these values can either be TRUE/FALSE or 1/0. Both are correct.
- invoices	- currency	The currency of the invoice is stored as locale as describe in the RFC 4646 (e.g. en_US, en_UK).
	- amount	The amount is a floating number that can have up to 4 decimals. The stored value is the amount of the invoice converted into the currency stored in the invoice.
- evaluation	- status	It defines the status of the S&F request. The possible values can be: - “create” - “pending” - “ongoing” - “done” - “cancelled”
- events	- timezone	The time zone of the event is stored as a string in the format “Continent/City”.
	- start_time - end_time	These “date” fields store a date in the calendar (MM/DD/YYYY) **and** a time (hh:mm:ss). On some database instance, this type is also called “datetime”

Table 3.5: Database entity additional information

Warning: This diagram has been made without considering the database type or version which is used in the project. It offers a generic model showing how the data are stored and related to each other.

3.4. Concurrency Viewpoint

3.5. Development Viewpoint

3.5.1. Symfony2 file system

The sources of SAFAPS SIM are installed on the server in `/var/www/safaps`. Below is the list of important files and folders that you might want to edit.

path	Description
`app/AppKernel.php`	It contains the list of bundles to load when an HTTP request is computed.
`src/ApiBundle/Controller/`	It contains the controller classes which are to be called when an HTTP request is received.
`src/ApiBundle/Resources/config/routing.yml`	It contains all routes the of the API the server is listening on. For each route, a controller class and a function inside it are defined.
`src/ModelBundle/Entity/`	It contains all entities with their ORM definitions and all entity repository classes of the application

Table 3.6: Important file and folder within source code

3.5.2. RabbitMQ

The server host for RabbitMQ is the Production Server (193.10.30.123) where it is installed using the following steps.

- Details about the installation of RabbitMQ for Symfony Framework and it's usage is available on the github https://github.com/php-amqplib/RabbitMqBundle
- Download the server from http://www.rabbitmq.com/install-windows.html
- Empty the Cache
- Run the Erlang Windows Binary File
- Run the installer rabbitmq-server-3.6.0.exe
- Customize the RabbitMQ Environment variables.

How to use:

All modification are in the folder /var/www/safaps.

All code about rabbitmq is in the folder ApiBundle.

Before sending a request you have to run the command in the server: rabbitmq-server & ("&" is used to do run the command in background, don't forget to stop it!)

You can use a script in /var/www/safaps/app/startRabbitMq.php to start rabbitMq server and start the Rpc.

To watch rabbitmq management interface, you can go to your web browser and use the url : 193.10.30.123/15672 (username: toto)

To Stop the server run the command : rabbitmqctl stop

Empty the cache if you see there is any modification or errors in your result.

3.5.3. Database Naming Convention

Hereafter is the naming convention and other rules adopted for the database implementation.

- Tables:
  - The table names are fully given in low camel case (lowercase letters with '_' as space between words).

Do	Don't
`organizations`	`Organizations` (first letter is uppercase)
`organization_invoices`	`organizationinvoices` (no '_' between words)

Table 3.7: Example of naming database tables

- Table IDs:
  - Each table representing an entity must have an ID. Only tables made for many-to-many relationships may be without any ID.
  - Entities' ID are "bigint" stored over 19 bits
  - Entities' ID are **unique**, "**non-nullable**", **primary keys**.
  - Entities' ID named 'id'
- Foreign keys:
  - Foreign keys are supposed to represent one-to-many or one-to-one relationships between two tables. For one-to-one relationship, either one or the other table is able to store the foreign key (but not both at once).
  - Foreign keys must be named `<fk_table_name>_<fk_attribute_name>`

Example:

Do	Don't
`organization_id`	`organizationid`
`evaluation_id`	`workfor`
`organization_id`	`administrator_organization_id`

Table 3.8: Example of naming foreign keys between database tables

3.6. Deployment Viewpoint

3.6.1. Production Environment

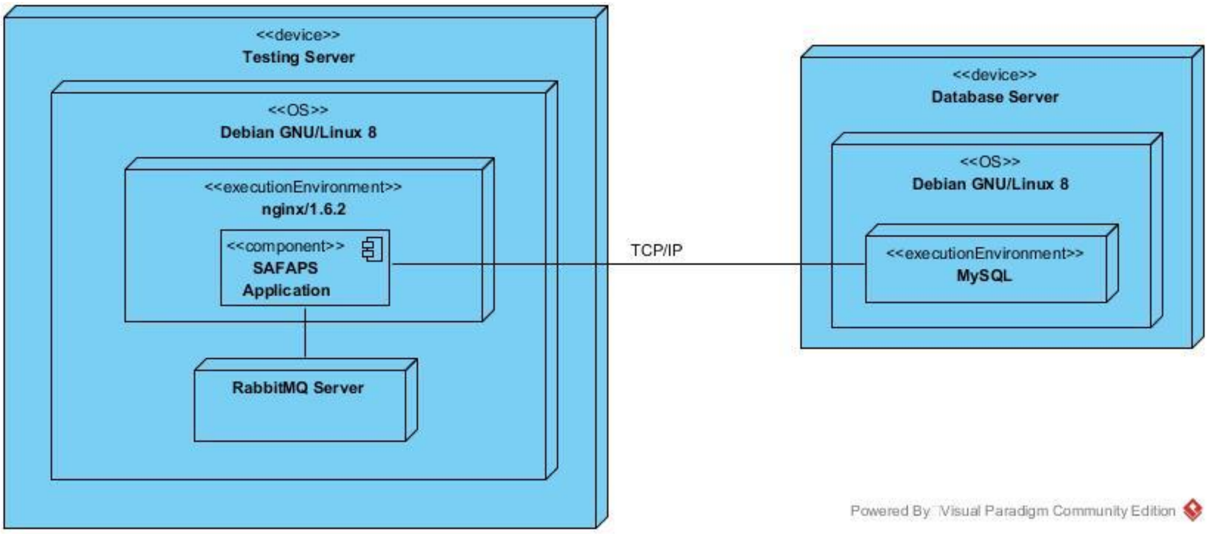


Figure 3.9: Deployment diagram of production environment

3.6.2. Testing Environment

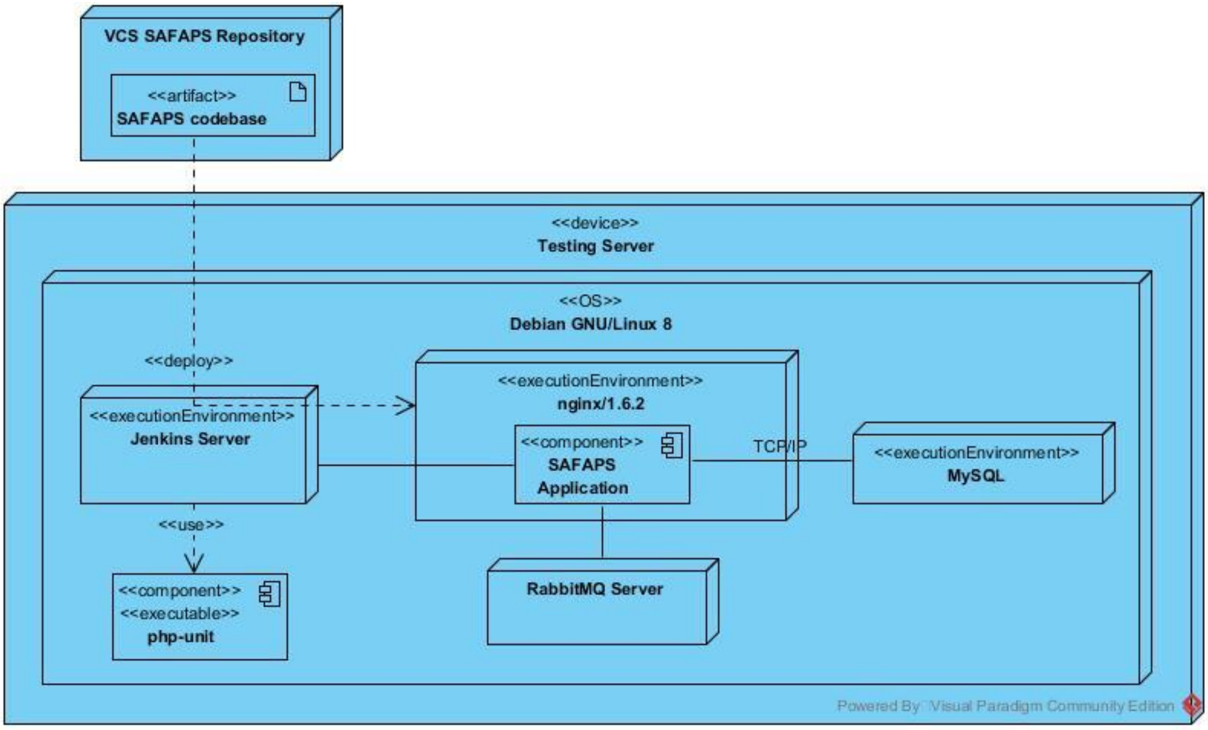


Figure 3.10: Deployment diagram of testing environment

3.7. Operational Viewpoint

4. Quality Property Summary

5. Important Scenarios

6. Issues Awaiting Resolution

7. Appendices